Challenges and Opportunities of Obtaining Performance from Multi-Core CPUs and Many-Core GPUs

Trista P. Chen and Yen-Kuang Chen*[+]

FX Palo Alto Laboratory* and Intel Corporation[+]

ABSTRACT

Multi-core processors represent a major development in computing technology. For example, Intel® Core™ 2 Quad processors, IBM Cell processors, and Nvidia GeForce 9800 GX2, are widely used. However, most applications struggle to make the best use of the power provided by many-core processors. Easy-to-use software tools are hard to find. Furthermore, it's not clear what changes need to be made to algorithms to fully utilize many-core CPUs or GPUs. In this paper, we try to offer a bird's eye view of the opportunities lying ahead in two folds: (1) software tools and (2) workload analysis. With good software tools and insightful workload analysis, software and algorithm developers can not only harness the power of many computing cores, but also innovate new algorithms that best utilize the many computing cores. New algorithms and applications are thus made possible with the computing power not available before.

Index Terms— GPU, CPU, many-core, multi-core

1. INTRODUCTION

The emerging multimedia applications demand more powerful computation than before. To name a few, multimedia data mining, 3D high-fidelity visualization, and image/video understanding, are among the most demanding workloads. It is shown that many emerging applications, e.g., recognition, mining, and synthesis, have found a variety of usage scenarios that require tera-floating-point-operations-per-second (tera-FLOPS) [1].

A single CPU core today can only provide giga-FLOPS of computation. Orders of magnitude of acceleration in computing power are needed. The introduction of multi-core CPUs and the promise of many-core GPUs [2] provide some light to such a need. As Moore's Law continues, we can expect more processing units to be available to us.

However, software programmers have discovered that it is easier to place many processing cores on a chip than to write efficient parallel codes for that many processing cores. Such trend was observed even in daily newspaper. The New York Times [3] noted that, *during the 80s, National Science Foundation tried to persuade the computer industry, but found little interest*; and *now the [multi-core] machines are here … to get around power wall*; however, "*newer chips with multiple processors require dauntingly complex software.*"

Additionally, existing serial codes need to be re-written to take advantage of the many-core computing power. The algorithms often need to be modified. This is because the best sequential algorithm is not necessarily the best parallel algorithm. As shown in Figure 1, one algorithm is faster than the other on a single-core processor. But it is on the other hand slower than the other on a 32-core processor.

This paper surveys opportunities for multimedia software and algorithm developers. The paper is organized as follows: Section 2 will describe why CPUs are going the direction of many-core, and how naturally parallel GPUs are evolving to general purpose computing. Section 3 will provide an overview of software tools for parallel computing on a variety of platforms (e.g., CPUs, GPUs, and Cell). Section 4 will briefly describe how workload analysis can help us obtain critical information for algorithm changes. Finally, we conclude the paper.

2. MULTI-CORE CPU AND MANY-CORE GPU

As more transistors are integrated into a single chip, the chip consumes more power. For example, from the mid-80s to the late 90s, the power consumption of Intel's

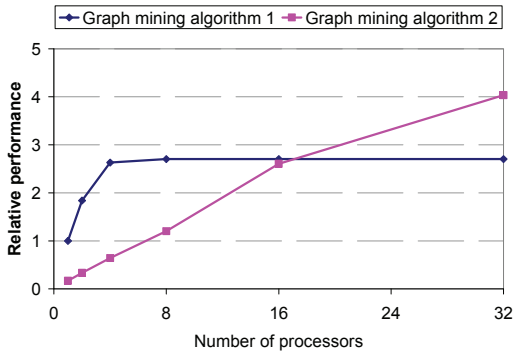

Figure 1: The best sequential algorithm is not the best parallel algorithm [4].

microprocessors follows the Moore's Law, doubling every two or three years [5], and reaching 20 watts/cm^2 from 1 watt/cm^2. Nonetheless, as power consumption approached the limit of sustainability, architects were forced to take a different direction [6].

To further increase performance without substantially increasing power consumption, parallel processing provides an alternative. For example, Intel® Core™ 2 Duo processors have lower thermal power consumption than Intel® Pentium® 4 processors. Multi-core processors represent a major development in computing technology. Nowadays, Intel® Core™ 2 Quad processors, IBM Cell processors, and Sun UltraSparc T1 processors, are widely used.

GPU was originally driven by game and graphics applications and is parallel in nature. Vertex and pixel processors work on massive amount of geometry and pixel data in parallel. Recent trend of GPU design has been on a unified architecture that allows for a single type of processing core. For example, Nvidia GeForce 9800 GX2 has 256 stream processors and is one order of magnitude from tera-FLOPS. Such design eliminates the need to estimate the distribution of workload characteristics and reduces its inefficiency when such an estimate is not accurate. In addition, more general computing, esp. signal processing, has been performed on GPUs [7][8][9] utilizing its near tera-FLOPS power, with its increasing programmability. More considerations of parallel computing can also be found at [10].

3. SOFTWARE TOOLS

Multi-core CPUs are currently in the market. Effectively using its multiple cores remains a challenge. Programmers have been struggled to obtain linear speed-ups with the number of cores. In addition, it is well known that the programmability of general-purpose computation on GPU (GPGPU) with hundreds of cores is not easy. This section provides an overview of the software tools that help the programmers to harness the many-core processing power.

3.1. Software Tools on CPU

In order to provide a portable and scalable model for developers of shared-memory parallel applications, a group of major computer hardware and software vendors got together in 90s to define a standardized Application Program Interface (API), called OpenMP [11]. Figure 2 shows an example of OpenMP codes in C/C++. It almost looks like a standard C code except for OMP constructs. The programmers do not need to explicitly manage the threads among cores. It uses a fork-join model to exploit thread-level parallelism for shared-memory systems. The programs begin as a single process: the master thread. The master thread executes sequentially until the first parallel region construct is encountered. The master thread then forks a team of parallel threads. The statements in the program that are enclosed by the parallel region construct are then executed in parallel among the various team threads. When the team threads complete the statements in the parallel region, they synchronize and terminate, leaving only the master thread at the join point.

```
void main()
{
  int blockSize = ...;
  for (int k=0; k<NUM_SAMS; k+=blockSize) {
   for (int i=0; i<NUM_SUPP_VEC; i++) {
    int lEnd_j = MIN(NUM_SAMS, k+blockSize);

    #pragma omp parallel for default(shared)
    for (int j=k ; j<lEnd_j ; j++) {
      result[j] += linear_kernel(&samples[j],
                                 NUM_VDIM, i);
    }
   }
  }
}
```

Figure 2. OpenMP parallel region example in C/C++

Through effective compiler and runtime library support, OpenMP can tackle the performance challenge. In [12], performance results of two multimedia applications using OpenMP demonstrate that we can effectively achieve good parallel performance by exploiting nested parallelism through the Intel compiler and runtime system support for OpenMP.

3.2. Software Tools on GPU

Fueled by the high volume gaming market, GPU computation capability has been growing tremendously over the past decade. However, most programs run on GPU are still highly specialized, geared towards graphics/gaming applications.

Brook for GPU [13] is one of the earlier attempts to make programming on GPU similar to C programming, and to harness GPU's parallel computation power. A Brook program consists of legal C code and extension to declare *streams* and denote given functions as kernels. It hides texture/pbuffer data management, graphics based constructs in CG/HLSK, and rendering passes. An example Brook program is shown in Figure 3.

Nvidia's Compute Unified Device Architecture (CUDA) [14][15] is a completely new architecture and programming model for general purpose computation on GPU. CUDA uses an extended C programming model.

```
kernel void updatepos (float2 pos<>,
                       float2 vel[100][100],
                       float timestep,
                       out float2 newpos<>) {
  newpos = pos + vel[pos]*timestep;
}

updatepos (positions, velfield, 10f, positions);
```

Figure 3. A Brook program example [13]

CPU C program

```
void add_matrix_cpu
     (float *a, float *b, float *c, int N)
{
    int i, j, index;
    for (i=0; i<N; i++) {
        for (j=0; j<N; j++) {
            index = i+j*N;
            c[index]=a[index]+b[index];
        }
    }
}

void main()
{
    ...
    add_matrix_cpu(a,b,c,N);
}
```

CUDA C program

```
__global__ void add_matrix_gpu
     (float *a, float *b, float *c, int N)
{
    int i=blockidx.x*blodkDim.x+threadidx.x;
    int j=blockidx.y*blodkDim.y+threadidx.y;
    index = i+j*N;
    if (i<N && j<N)
        c[index]=a[index]+b[index];
}

void main()
{
    ...
    dim3 dimBlock (blocksize,blocksize);
    dim3 imGrid (N/dimBlock.x,N/dimBlock.y);
    add_matrix_gpu<<<dimGrid,dimBlock>>>(a,b,c,N);
}
```

Figure 4. Comparison of CPU C and CUDA C

CUDA programmers let hardware thread manager handles threading automatically instead of programmers themselves. Such model makes deadlocks among threads impossible. Programmers need to focus only on data decomposition among thread processors. An example CUDA program is shown in Figure 4, where multiple levels of parallelism are utilized: thread (identified by threadIdx), thread block (identified by blockIdx), and grid of thread blocks.

3.3. Software Tools: across multiple platforms

While OpenMP is suitable for CPU platforms, Brook and CUDA are suitable for GPU platforms, PeakStream (acquired by Google in June 2007) [16] and RapidMind [17] are tools for across different platforms.

PeakStream is a data-parallel stream programming model for many-core processors. It is portable, aiming to run across vendors and processor generations. It also means to be interoperable to leverage existing libraries and tools as MPI and gcc compiler. PeakStream programming model uses API instead of introducing a new language for API's relatively easier adoption. Data are expressed as arrays of 32 or 64 bit floating point numbers. Operator overloading coverts operators into data parallel operators. The API looks like Intel MKL, Fortran, and Matlab functions. Functions "make" and "write" move data onto the cores for processing. Stream arrays are opaque. Data is copied back to the system memory with "read" calls. PeakStream Platform includes a virtual machine and its just-in-time (JIT) compiler. At run time, PeakStream's virtual machine intercepts the special function calls embedded in the x86 binary. Such dynamic compilation facilitates binary portability.

Similar to PeakStream, RapidMind uses hardware abstraction layer and JIT compilation to optimize parallel codes. Unlike PeakSteam, programmers of RapidMind can define their own functions in addition to the provided API. RapidMind's abstraction layer automatically distributes tasks among cores, either homogeneous cores or heterogeneous cores. Hence, deadlock threads and thread-synchronization problems are eliminated like Nvidia CUDA. The JIT compilation avoids code re-writing hassle when the same code is to run on a different hardware architecture, for example, from 2 to 4 cores. In addition, dynamic compilation allows run-time optimization based on system feedbacks. Some workload, e.g., Black-Scholes model, can even perform better than linear speed-ups when the number of cores increases [17].

Ambric [18] takes a different approach than all the software tools mentioned above. They started by considering the programming tool, then developed hardware architecture that met the software model. It is however for embedded solutions rather than general purpose computing. Such methodology for parallel computing might be a solution on the general purpose computing platform.

4. WORKLOAD ANALYSIS

With the software tool, application developers can still feel daunted with the possibility of great amount of re-writing for the existing codes. Furthermore, application developers may face challenges in finding the right algorithms for many-core CPUs or GPUs. The return of developing a brand new algorithm for many-core processors is unclear even if the application developers decide to do so. Workload analysis provides the key to these problems.

First, we must examine the inherent coarse-grained and fine-grained, data-domain and functional-domain parallelisms in the workload. For example, computer vision algorithms can be categorized as low-level image processing and high-level statistical analysis. Data-domain parallelization partitions the data into independent pieces, e.g., blocks of pixels. Functional-domain parallelization decomposes the computation into stages, e.g., gradient computation, hysterisis testing, and so on.

Second, we evaluate the parallel performance on the current hardware and projected future hardware. We should characterize the performance bottlenecks (including memory behavior) in detail. For example, in [19], we show that the original implementation of the serial operation in Canny Edge detection [20] prevents the algorithm from scaling well in a multi-core system. By parallelizing the hysteresis-testing portion of the algorithm, we can have a 5x performance gain on a 32-core processor.

Such methodology is applicable to various computing platforms, such as GPUs and the Cell. For example,

- In [21], Lloyd et al. show a comprehensive work from workload analysis to algorithm changes on many-core GPUs. After surveying in-place and out-of-place algorithms, the authors decided to choose an out-of-place algorithm based on two good reasons: (1) Stockham FFT has better memory access pattern, and (2) texture cannot be read and written at the same time.
- In [22], Wu et al. describe the steps to improve the performance of their BSB implementation on Cell processors, e.g., matrix shuffle, loop unrolling, double buffering, etc. The result is very impressive. Their implementation can achieve 70% of the peak performance. When they can utilize 6 SPEs simultaneously, the performance won't be easily matched by a workstation.

5. CONCLUSION

Processors with multiple cores are going main stream [23]. This not only applies to general-purpose CPUs, but also to GPUs and even System-on-a-chip (SoC). This implies that we can enable many emerging multimedia applications that were not possible when the computing power was unavailable. However, how to get the best performance out of many-core CPUs and GPUs is a challenge, in particular, how to develop software and algorithms for emerging multimedia applications.

The opportunity lies in two folds: (1) software tools and (2) workload analysis. With better software tools, software developers can easily make the best use of the power of the many computing cores. Additionally, analyzing the workloads and considering architectural style can help us adapt, choose, and develop algorithms.

6. REFERENCES

[1] Y.-K. Chen, J. Chhugani, P. Dubey, C. J. Hughes, D. Kim, S. Kumar, V. W. Lee, A. D. Nguyen, and M. Smelyanskiy. "Convergence of Recognition, Mining, and Synthesis Workloads and its Implications", *Proceedings of IEEE*, vol. 96, no. 5, pp. 790 – 807, May 2008.

[2] High Performance and Supercomputing with NVIDIA Tesla: http://www.nvidia.com/object/tesla_computing_solutions.html.

[3] J. Markoff, "Faster Chips are Leaving Programmers in Their Dust," *The New York Times*, Dec. 2007.

[4] G. Buehrer, S. Parthasarathy, and Y.-K. Chen, "Adaptive Parallel Graph Mining for CMP Architectures," *IEEE International Conference on Data Mining*, pp. 97-106, Dec. 2006.

[5] F. Pollack, "New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies," *IEEE/ACM Int'l Symp. on Microarchitecture*, 2001.

[6] R. M. Ramanathan, M. Agan, A. Daniel, and P. A. Correia, "Intel Energy-Efficient Performance---Performance Made Energy Efficient Through New Technological Leaps," *Intel whitepaper*, available from http://download.intel.com/technology/eep/overview-paper.pdf.

[7] GPGPU: http://www.gpgpu.org/.

[8] M. D. McCool, "Signal Processing and General-Purpose Computing on GPUs", IEEE Signal Processing Magazine, May 2007, pp. 109-114.

[9] J. Fung and S. Mann, "Using Graphics Devices in Reverse with OpenVidia: GPU-based Image Processing and Computer Vision", *ICME 2008*, Hanover, Germany, June 2008.

[10] K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, S. Williams, and K. Yelik, "The Landscape of Parallel Computing Research: A View from Berkeley", *Technical report*, EECS Department, University of California at Berkeley, UCB/EECS-2006-183, December, 2006.

[11] OpenMP: http://www.openmp.org/

[12] X. Tian, J. Hoeflinger, G. Haab, Y.-K. Chen, M. Girkar, S. Shah, "A Compiler for Exploiting Nested-Parallelism in OpenMP Programs," *Parallel Computing Journal*, vol. 31, no. 10-12, pp. 960-983, Oct. 2005.

[13] Brook for GPU: http://graphics.stanford.edu/projects/brookgpu/.

[14] Nvidia CUDA Homepage: http://developer.nvidia.com/object/cuda.html.

[15] T. R. Halfhill, "Parallel Processing with CUDA", *Microprocessor Report*, January 28, 2008.

[16] Tom R. Halfhill, "Number Crunching with GPUs", *Microprocessor Report*, October 2, 2006.

[17] T. R. Halfhill, "Parallel Processing for the x86", *Microprocessor Report*, November 26, 2007.

[18] Tom R. Halfhill, "Ambric's New Parallel Processor", *Microprocessor Report*, October 10, 2006.

[19] T. Chen, D. Budnikov, C. Hughes, and Y.-K. Chen, "Computer Vision on Multi-Core Processors: Articulated Body Tracking," ICME 2007, Beijing, China, July 2007.

[20] J. Canny, "A Computational Approach to Edge Detection", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol 8, No. 6, Nov 1986.

[21] D.B. Lloyd, C. Boyd, and N. Govindaraju, "Fast Computation of General Fourier Transforms on GPUs", *ICME 2008*, Hanover, Germany, June 2008.

[22] Q. Wu, P. Mukre, R. Linderman, T. Renz, D. Burns, M. Moore, and Q. Qiu, "Performance Optimization for Pattern Recognition using Associative Neutral Memory", *ICME 2008*, Hanover, Germany, June 2008.

[23] W. Gibbs, "A Split at the Core," *Scientific American*, Nov 2004.